NPTSN: RL-Based Network Planning with Guaranteed Reliability for In-Vehicle TSSDN

Weijiang Kong, Majid Nabi, Kees Goossens
Department of Electrical Engineering, Eindhoven University of Technology, the Netherlands
{w.kong, m.nabi, k.g.w.goossens}@tue.nl

***Abstract*—To achieve strict reliability goals with lower redundancy cost, Time-Sensitive Software-Defined Networking (TSSDN) enables run-time recovery for future in-vehicle networks. While the recovery mechanisms rely on network planning to establish reliability guarantees, existing network planning solutions are not suitable for TSSDN due to its domain-specific scheduling and reliability concerns. The sparse solution space and expensive reliability verification further complicate the problem. We propose NPTSN, a TSSDN planning solution based on deep Reinforcement Learning (RL). It represents the domain-specific concerns with the RL environment and constructs solutions with an intelligent network generator. The network generator iteratively proposes TSSDN solutions based on a failure analysis and trains a decision-making neural network using a modified actor-critic algorithm. Extensive performance evaluations show that NPTSN guarantees reliability for more test cases and shortens the decision trajectory compared to state-of-the-art solutions. It reduces the network cost by up to 6.8x in the performed experiments.**

***Index Terms*—TSD, SDN, Recovery, Network Planning.**

I. Introduction

Time-Sensitive Networking (TSN) is considered a promising solution for future In-Vehicle Networks (IVN). Leveraging the standardized Ethernet solutions, TSN offers bandwidth that constantly grows with the technology. Meanwhile, it introduces amendments to provide real-time and reliable services for automotive applications, which is beyond the scope of the conventional Ethernet. In IEEE 802.1Qbv standard [1], Time Aware Shaping (TAS) is specified to enable Time-Triggered (TT) transmission which offers deterministic service for critical control applications (e.g., flows for steering and braking).

Another essential requirement for IVN is to provide guaranteed reliability for these safety-critical flows. The reliability of the flows is usually specified by the ISO 26262 standard for Road Vehicles - Functional Safety [2]. Based on the severity, exposure, and controllability of the failures, the standard defines four Automotive Safety Integrity Levels (ASIL), denoted as ASIL A-D from least to most critical. Additionally, the standard defines ASIL decomposition to build high-ASIL functionalities with redundant but low-ASIL components to reduce cost. TSN addresses the reliability concerns in the IEEE 802.1cb standard for Frame Replication and Elimination for Reliability (FRER) [3], which specifies that frames can be replicated and forwarded via redundant paths. FRER protection can be designed as an ASIL decomposition problem [4]. However, since it relies on static redundancy, FRER is expensive and requires duplicating the entire network in the worst case.

This research was supported through PENTA project HIPER 181004 on high performance vehicle computer (HPVC) and communication system for autonomous driving.

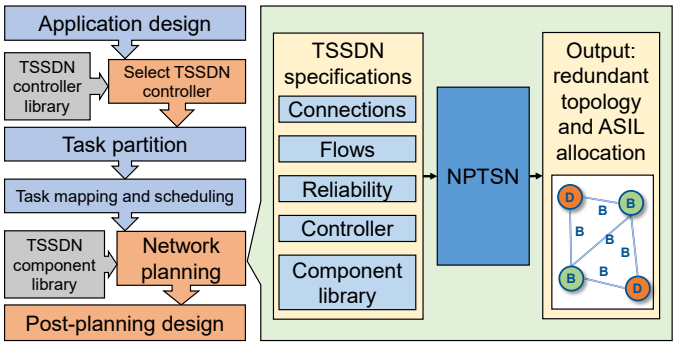


Fig. 1: Design flow of in-vehicle TSSDN

As the standard gradually matures, there is a strong trend for TSN to evolve towards Time-Sensitive Software-Defined Networking (TSSDN) [5], which enables adaptive network management by decoupling the TSN control and data plane. In networks with limited scale and sufficient global information (e.g., the status of all switches), TSSDN demonstrates promising flexibility [6], reliability [7], and cost-efficiency [8]. Thus, IVN is an ideal use case for it. An important application of TSSDN is run-time recovery. With its rapid reconfiguration capability, TSSDN can adaptively reroute the flows to bypass failures, and arbitrary recovery strategies can be programmed into its controller software, which inspires the innovation of recovery mechanisms. Compared to FRER, run-time recovery achieves the reliability goal with lower redundancy cost and offers better protection against high-order failures [9].

Run-time recovery has recently received wide research attention. But existing techniques offer reliability in a “best-effort” manner, i.e., they attempt to maximize the recovery capability on given networks that are assumed to be sufficiently redundant. Otherwise, the recovery mechanism by itself cannot provide reliability guarantees. Therefore, TSSDN with run-time recovery relies on the design flow illustrated in Fig. 1. The TSSDN controller, which has been widely researched, can be reused from the existing design. Then, tasks from both the applications and the controller are partitioned, mapped to resources, and scheduled. Later in the network planning stage, reliability guarantees are established. The TSSDN network planning tools must design the topology to provide the necessary redundancy for recovery and specify the network components (links and switches) in terms of ASIL. It is followed by the post-planning design where the designers address other functional safety

concerns, such as ensuring the partitioning property [10] via policing and introducing system-level protection mechanisms.

Current network planning tools are developed for general network planning problems and are not suitable for in-vehicle TSSDN. There are three major challenges. First, TSSDN involves domain-specific requirements. The component ASIL must be allocated properly. Moreover, TSSDN must provide strict latency guarantees for its flows. While general network planning approaches consider a failure survivable if the residual network remains connected [11], TSSDN recovery additionally requires scheduling the flows on the residual network. Thus, network planning must consider the dynamic recovery behavior with the awareness of the domain-specific schedulability requirement. The second challenge is the sparse solution space of the problem. Due to the reliability requirement, TSSDN prefers redundant topologies. But switches have a limited number of ports, which restricts the connectivity of the networks. Thus, although the number of topologies is huge, only a limited fraction of it is valid for TSSDN. Typical optimization techniques such as Cross-Entropy Method (CEM) [12] explore the solution space stochastically to capture its latent pattern, then improve the exploration strategy. However, for TSSDN, a completely random exploration may seldomly acquire feasible solutions, which makes the strategy update inefficient. Finally, reliability verification is expensive. The reliability analysis applied to every explored topology may involve checking the recovery behavior for thousands of failure scenarios. Hence, efficient exploration, which finds good solutions with fewer attempts, and fast failure analysis are strongly preferred.

In this paper, we propose NPTSN, a network planning solution for in-vehicle TSSDN based on deep Reinforcement Learning (RL). Our insight is that any domain-specific requirement can be represented by the environment dynamics of RL, and the underlying pattern of the recovery behaviors can be captured by deep neural networks to perform efficient exploration. Given the component library and the flows, NPTSN iteratively constructs the TSSDN with an intelligent network generator. To efficiently explore the sparse solution space, we propose a knowledge-based action generation, which actively prunes away invalid actions, referred to as the Survival-Oriented Action Generation (SOAG). In every iteration, NPTSN evaluates the reliability of the TSSDN with a failure analysis. We define stateless recovery behaviors and design the action space to limit the failure scenarios being checked. Then, the SOAG proposes actions and masks that contribute to the network reliability based on the failure analysis. Thus, feasible solutions are more likely to be explored. The RL-based decision maker selects the action that modifies the TSSDN towards its reliability goal. We use a novel encoding method to formulate the TSSDN and the actions into its observation (input) in such a way that the neural networks inside the RL-based decision maker can be trained stably on the dynamic action space. We train the neural networks using a modified actor-critic algorithm that integrates the knowledge-based generation with the neural network-based generation process. Our experiments show that compared with existing solutions, NPTSN ensures reliability guarantees in more test cases while reducing the network cost by up to 6.8x.

The paper is structured as follows. Section II introduces our system model and the TSSDN planning problem. Section III provides an overview of the NPTSN architecture. Section IV discusses the intelligent network generator including the SOAG and the RL-based decision maker. Section V introduces the failure analyzer. We present the evaluation results in Section VI, introduce the related works in Section VII, and conclude the paper in Section VII.

II. System Model

A TSSDN can be modeled as a 5-tuple (G^t, G^f, B, FS, FI) in which G^t is the topology, G^f is the failure scenario, B is the base period, and FS (FI) is the specification (state) of the flows. The behavior of the TSSDN upon failures can be generally represented by the Network Behavior Function (NBF) denoted as Φ. In this section, we first discuss our TSSDN model and then introduce its network planning problem.

A. TSSDN Model

The topology of a TSSDN can be represented as an undirected graph G^t. The vertices of this graph (V) consist of end stations (V_{es}) and the switches (V^t_{sw}) selected during the network planning stage to connect the end stations. $deg(v)$ denotes the degree of a vertex $v \in V$. The edges E^t represent links. Every undirected edge $(u, v) \in E^t$ denotes the bidirectional connection between u and v. Note that we consider links with uniform bandwidth, which is a typical setup for TT transmission. NPTSN plans spatial redundancy to handle random failures that permanently affect network components. They can be caused, among others, by aging, electromigration, and thermal cycling of the electronic components, whose probability can be predicted with reasonable accuracy. We assume the switches and links to be fail-silent. A failure scenario G^f can be represented by a subgraph of G^t, whose vertices V^f (edges E^f) denote the malfunctioned nodes (links). Note that when a link fails, connections are closed for both directions; When a switch fails, any link attached to it cannot be used.

Real-time applications in TSN often generate frames to be transmitted periodically. Such frame series are referred to as flows. NPTSN considers TT flows, which are safety-related flows for control purposes. To initiate a flow, applications inform the central user configuration [13] about its specification, which consists of the source, destinations, period, and frame size. The specification of all TT flows is denoted by FS. We assume FS stays constant since the beginning of the network because safety-critical applications in vehicles seldomly change on the run. Based on FS, the central network configuration [13] adaptively updates the network to handle failures.

TSN performs priority-based queuing, i.e., frames are buffered in different egress queues based on their priorities. TAS [1] specifies that the queues are controlled by the corresponding transmission gates, which cyclically execute the schedule defined in the gate control list based on a globally synchronized clock. The period of the global TAS schedule is the base period B. It is determined before the network starts and never changes on the run [5]. To recover flows from failures, the

central network configuration re-schedules the flows resulting in new flow states FI'. FI consists of the flow paths, an ordered set of links through which the flow will be forwarded, as well as the time slots reserved on each link. TT flow scheduling is a well-studied problem, on which run-time recovery relies to restore flows from failures.

B. TSSDN Recovery Behavior

TSSDN adaptively changes its flow states in response to failures, which essentially involves re-scheduling the TT flows on the residual network. Because the recovery mechanisms evolve constantly, NPTSN models the recovery behavior with a general notion to cover a wide range of recovery mechanisms. Arbitrary recovery behavior can be represented by a stateful NBF $\Phi^s : G^t, G^f, B, FS, FI \mapsto FI', ER$, in which G^t, B, FS, FI are the topology, base period, flow specification, and flow state of the network before failures. G^f is a subgraph of G^t representing the failures encountered (by both nodes and links). The NBF models the process to re-schedule flows to bypass failures, in which the bandwidth and timing guarantees of the flows must be re-established. Without flow-level redundancy, the recovery succeeds if all TT flows are schedulable. The output FI' is the new flow state after recovery. If recovery fails, i.e., there are flows whose bandwidth and timing guarantees cannot be re-established, ER returns the error message. TSSDN can propagate the error message to the applications to perform system-level service degradation [9]. ER consists of node pairs. Each node pair $(s, d) \in ER$ ($s, d \in V^t_{es}$) represents a pair of source and destination nodes that FI' fails to recover, i.e., $ER = \emptyset$ if recovery succeeds. When G^f is an empty graph, we defined $FI' = FI_0$ for any input FI. FI_0 is the initial flow state on topology G^t which is often generated by offline approaches [7]. The corresponding ER_0 is the source and destination pairs between which the bandwidth and timing guarantee of the flows cannot be established. NBF is a deterministic function once the TSSDN controller is selected and it can be obtained via network simulation.

Verifying the stateful NBF under multi-point consecutive failures (e.g., link 1 fails first, then link 2 fails) is expensive in terms of computation complexity because the flow state after recovery FI' depends on FI, which is the flow states before recovery. To consider the sequence with which the failures may occur, an n-point failure requires verifying $n!$ flow states. Thus, NPTSN requires the NBF to be stateless. The stateless NBF is defined as $\Phi : G^t, G^f, B, FS \mapsto FI', ER$. It means the flow states after recovery do not depend on the previous flow state. Thus, every failure scenario leads to only one flow state. [14] is an example of such stateless recovery scheme. Some other recovery schemes are stateful mostly because they compare FI with G^f to only reschedule the disrupted flows [7], [9]. A minor modification can make their NBF stateless: instead of using the current FI as the reference, the new flow state can be computed based on the initial flow state FI_0 ($\Phi^s(G^t, G^f, B, FS, FI_0)$). This does not impact the recovery of single-point failures. But for multi-point consecutive failures, potentially more flows will be reconfigured. Thus, the recovery process can become more expensive.

TABLE I: A component library with normalized cost.

Switch Library					Link library		
ASIL	Cost w.r.t. # of ports			Failure prob.	ASIL	Cost/unit length	Failure prob.
	4-port	6-port	8-port				
A	8	10	16	10^{-3}	A	1	10^{-3}
B	12	15	24	10^{-4}	B	2	10^{-4}
C	18	22	36	10^{-5}	C	4	10^{-5}
D	27	33	54	10^{-6}	D	8	10^{-6}

C. Network Planning Problem

Network planning receives an undirected graph of possible connections G^c, whose vertices V^c contain the end stations to be connected (V_{es}) and the set of optional switches (V^c_{sw}). Its edges E^c represent the set of optional links whose lengths $len(u, v), \forall (u, v) \in E^c$ are the distances of the connection. The available connections depend on the length of the network cables, the placement of the nodes, and the wiring constraints inside the vehicles. In the ideal case, G^c contains the complete set of connections ($E^c = \{(u, v) | \forall u \in V^c_{sw}, v \in V^c, u \neq v\}$). But in reality, especially for large networks, directly connecting nodes on different sides of the diameter is infeasible. Thus, E^c is typically a subset of the complete connections. The output TSSDN topology G^t (Section II-A) is a subgraph of G^c that connects the end stations with a subset of the optional links and switches, i.e., $V^t_{sw} \subseteq V^c_{sw}$, $E^t \subseteq E^c$. Note that the end stations are defined by the applications so their cost, ASIL, and the maximum number of ports are assumed given.

Network planning involves choosing the links and switches from the component library. Note that small switches can be combined into large switches. These combined switches can also be included in the library to enable more port options. An example library is shown in TABLE I. We will discuss how it is obtained in Section VI. Besides specifying the network topology, NPTSN also specifies ASIL for links $ASIL_{u,v}, \forall (u, v) \in E^t$ and switches $ASIL_v, \forall v \in V^t_{sw}$. Then, switches with the fewest ports (lowest cost) can be selected while links can be chosen directly from the library. More specifically, the cost of a switch $v \in V^t_{sw}$ is a function of its degree and ASIL $csw(deg(v), ASIL_v)$. More ports and higher ASIL lead to higher costs. Also, to ensure that feasible switches exist, the topology must constrain the switch degrees, e.g., the maximum switch degree allowed in the example above is 8. Meanwhile, the cost of a link $(u, v) \in E^t$ is determined by its ASIL and cable length $clk(ASIL_{u,v}, len(u, v))$. NPTSN supports arbitrary cost functions for links. Eq. 1 computes the cost of a network G^t whose ASIL has been specified.

$$\begin{aligned} network\ cost = &\sum_{v \in V^t_{sw}} csw(deg(v), ASIL_v) \\ &+ \sum_{(u,v) \in E^t} clk(ASIL_{u,v}, len(u, v)) \end{aligned} \tag{1}$$

The component failure probability depends on its ASIL and can be denoted as a function $cfp(ASIL)$. Eq. 2 gives the probability that a failure G^f happens.

$$\begin{aligned} &probability\ of\ failure\ G^f \\ &= \prod_{v \in V^f} cfp(ASIL_v) \times \prod_{(u,v) \in E^f} cfp(ASIL_{u,v}) \end{aligned} \tag{2}$$

The ISO 26262 standard [2] requires eliminating single or dual-point failures. High-order failures can be considered safe faults which do not significantly impact the reliability objective. We generalize this for TSSDN with dynamic redundancy. The *network level reliability guarantee* is specified as *the maximum probability of safe faults*, denoted as R. Network planning must provide sufficient spatial redundancy so that the network can be recovered from any failure with a probability larger than R. Thus, NPTSN is not restricted to certain failure orders. But analyzing extremely rare failures will require high computation complexity. Without redundant flows, the network fails if an end station fails. Thus, end stations naturally require high reliability so that their failures are considered safe faults.

In summary, network planning receives the graph of possible connections G^c, the base period B, the flow specifications FS, the stateless NBF Φ, the reliability goal R, and the component library as input. Its output includes the network topology G^t, as well as the ASIL allocated to its links ($ASIL_{u,v}, \forall (u,v) \in E^t$) and switches ($ASIL_v, \forall v \in V^t$). The objective of network planning is to minimize the network cost subject to the degree constraint and the reliability guarantee.

III. Overview of NPTSN

In TSSDN, the correlation between topology and reliability is determined by the complex network behavior, which makes efficient optimization difficult. Meanwhile, traditional approaches typically rely on heuristics that assume certain properties of the recovery mechanism. They might require frequent adjustment as new recovery mechanisms emerge. Thus, NPTSN considers a wide range of recovery mechanisms through the NBF abstraction and performs efficient optimization with RL.

Fig. 2 illustrates an overview of NPTSN. It consists of two major components, an intelligent network generator and a failure analyzer. The network generator iteratively proposes TSSDN solutions and the failure analyzer verifies the reliability requirement to provide feedback to the network generator. The process starts with an empty TSSDN consisting of end stations only (no link or switches). Within the network generator, the SOAG dynamically generates a coarse-grained action space consisting of switch upgrade and path addition actions, which prunes away invalid actions. These actions aim to resolve the non-recoverable failures found during the last failure analysis so they potentially improve the reliability of the proposed TSSDN. The RL agent makes the decision to execute one of the provided actions, resulting in an updated TSSDN solution. Then, the failure analyzer checks if the network can survive all non-safe faults by simulating the NBF. If the reliability requirement is met, the solution is recorded and the TSSDN is reset to empty for the next attempt.

In RL, each decision of action is referred to as a step. The RL agent decides the next action using deep neural networks. The neural networks are trained for a specified number of iterations (epochs). In every epoch, the RL agent first runs the current neural networks for a specified number of steps and then performs a gradient update to minimize the cost of the generated TSSDN. When the training finishes, the best TSSDN solution discovered (during all epochs) is the result of the network planning problem.

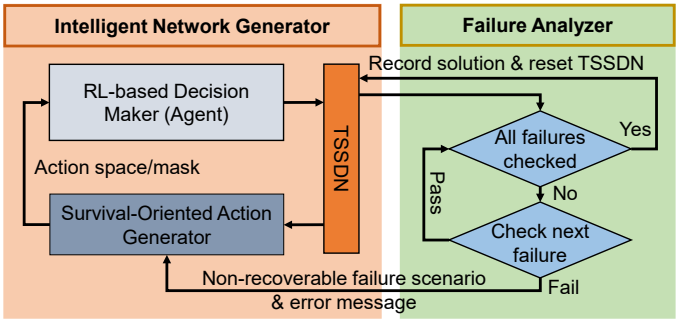


Fig. 2: An overview of NPTSN

IV. Intelligent Network Generator

In this section, we first provide a brief introduction of RL then discuss the intelligent network generator, which consists of the SOAG and the RL-based decision maker.

A. Introduction to RL

RL is a delicate approach for multi-step decision-making problems, which are challenging due to computation complexity. It models the decision process as an agent interacting with the surrounding environment to gain rewards. The agent observes the environment and "learns" its latent pattern to improve its strategy. The environment then changes its state according to the actions and provides rewards to the agent based on the decision problems. RL has several advantages. It can directly represent the desired properties via environment dynamics and perform active exploration of the solution space, obviating human efforts in designing and tuning heuristics [15]. Moreover, deep RL leverages deep neural networks to identify the impact of the actions even if the corresponding reward is delayed [16]. Thus, it often outperforms human experts, which tend to make greedy decisions in complex problems.

The RL problem can be formulated by Eq. 3 [17]. $\theta(\pi)$ represents the policy, which is the strategy to select actions, parameterized by π. τ is the trajectory obtained through the policy, which consists of sequences of states (s_t) and actions (a_t), i.e., $\tau = \{..., s_t, a_t, s_{t+1}, a_{t+1}, ...\}$. γ_t is the discounted reward obtained by performing action a_t at state s_t. RL aims at finding the optimal policy that maximizes the reward expectation. Deep RL represents the policy with neural networks, which takes the current state as input (observation) and decides the next action. The parameters of the neural networks can be optimized through gradient-based methods.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \theta(\pi)} \left[\sum \gamma_t(s_t, a_t) \right] \tag{3}$$

NPTSN formulates the TSSDN planning problem as an RL problem. Section IV-B introduces our NBF-knowledge-based action generation. Section IV-C discusses the design regarding reward, neural networks, encoding, and training algorithms.

Algorithm 1: Compute path addition actions and masks

input : connection graph G^c, topology G^t, failure G^f, error message ER, parameter K;
output: path actions and masks

1 Randomly select $(s, d) \in ER$
2 $G = G^c$ // V (E) is nodes (edges) of G
3 $G.remove\_nodes(V^f \cup (V^c_{sw} \setminus V^t_{sw}))$
4 $G.remove\_edges(E^f)$
5 $PathAction = K\_Shortest\_Path(G, s, d, K)$
6 $m_i = 1, \forall m_i \in PathActionMask$
7 **for** $p_i \in PathAction$ **do**
8 $\quad G' = (V, E \cup p_i)$
9 $\quad$ **if** $(\exists v \in V'_{sw}, deg(v) > MaxSWDegree) \vee$ $(\exists v \in V'_{es}, deg(v) > MaxESDegree)$ **then**
10 $\quad\quad m_i = 0$
11 $\quad$ **end**
12 **end**

B. Survival-Oriented Action Generator

A good action design aids the RL agent to explore good solutions. We design the actions in NPTSN to tune paths instead of individual links because paths are the minimum connectivity from the perspective of the flows. To avoid enumerating all possible paths, NPTSN dynamically generates paths in the action space based on the feedback from the failure analysis.

The SOAG takes two inputs from the failure analyzer: G^f which is an unrecoverable failure scenario and ER which is the error message under that failure. The goal of SOAG is to generate the actions that can potentially help the TSSDN to survive the failure G^f. The dynamic action space consists of $|V^c_{sw}|$ switch upgrade actions and K path addition actions. To disable invalid actions, every action is associated with a bit in the action mask. The RL agent is designed to only select the actions masked by one. Note that K is an adjustable parameter. Higher K indicates a larger coverage of the solution space while lower K leads to faster action generation and smaller neural networks. The actions are as follows.

- **Switch upgrade**: switch upgrade actions add new switches with the lowest ASIL or increases the ASIL of existing switches. When a new switch is added, its ASIL is set to A. Otherwise, if a switch has been added and its ASIL is lower than D, the corresponding action increases its ASIL by one level, e.g., A to B. ASIL-D switches cannot be upgraded so their masks are set to zero.
- **Path addition**: path addition actions add new paths computed by Algorithm 1 to the networks. The SOAG attempts to connect one source and destination pair randomly selected from the error message each time (line 1). Every path is a set of links from the source to the destination and it can only traverse switches that have been previously added. Algorithm 1 obtains optional paths via the k shortest path algorithm [18] and disables masks for paths that violate the degree constraint (line 6-12).

During both switch upgrades and path addition, we ensure that the ASIL of every link (u, v) equals the lowest ASIL of the adjacent vertices u and v. There are three major reasons. First, further increment of the link ASIL increases the network cost but has a negligible impact on system reliability. Second, introducing extra actions to tune link ASIL leads to a huge action space that is not scalable. Finally, as we will discuss in Section V, it significantly simplifies the failure analysis.

An advantage of the dynamic space in NPTSN is that its actions are more likely to improve the network toward a valid solution. In contrast, actions that add individual links [16] require a series of good decisions to achieve a similar improvement and it is more difficult to discover a good policy during the exploration phase. Besides this, our coarse-grained actions shorten the decision trajectory, i.e., unnecessary connectivity is less likely to be introduced and less effort will be spent on the (costly) failure analysis. Note that NPTSN constructs TSSDN monotonically, i.e., switch degradation and link removal actions are not allowed. Any networks achievable with these additional actions can be obtained without them. Moreover, they cause a negative impact on algorithmic scalability and make it difficult to determine when the exploration should stop.

Note that an alternative option to generate the path addition actions is to find K paths satisfying the degree constraint with their masks all set to one. This provides more actions for the RL agent. However, in case valid paths do not exist, it would exhaustively check all paths between the source and destination pair, which is not acceptable because of the execution time.

C. RL-Based Decision Maker

The RL-based decision maker is based on the actor-critic algorithm [19], which requires two neural networks: an actor network to represent the policy and select actions, and a critic network to estimate the value function of the selected actions. This section explains the RL-based decision maker in detail.

Reward Design: the cost objective is encoded by the reward. The reward of every action equals the previous network cost minus the network cost after the action is taken. Hence, the sum of the reward when a valid solution is found approximately equals the negative network cost (the discount factor is typically less than 1 to avoid an infinite reward sum). The agent maximizes the reward to minimize the network cost. We scale down the reward into [-1, 0) by multiplying it with a reward scaling factor, which is a widely used technique to avoid saturation and inefficiency problems [20]. If a valid solution has not been found when there are no valid actions (all action masks are zero), the final reward is subtracted by one as an extra penalty. Note that giving positive rewards for valid solutions is an alternative and leads to similar training results.

Neural Network Architecture: Graph Convolutional Networks (GCN) [21] is a well-developed neural network architecture to efficiently extract information from graph-based data. Its motivation is to propagate messages between adjacent vertices in every GCN layer. By concatenating multiple such layers, global features can be extracted into a low-dimension graph embedding vector. More precisely, the layer-wise propagation rule in GCN can be represented by Eq. 4 [21], where H^l is the output matrix of layer l. A is the adjacency matrix of the input graph. The identity matrix I adds the self-connection to A. D is

Algorithm 2: Actor-critic algorithm in the RL-based decision maker

input : connection graph G^c; TSSDN specifications B, FS, Φ, R; component library; SOAG Parameter K
output: the best TSSDN found

Initialize neural networks with randomized parameters
for $i \in \{1, ..., maxepoch\}$ **do**
  `// explore current policy`
  Initialize $TSSDN$ (topology with end stations only), $Action$, and $Mask$; clear RL Buffer
  for $j \in \{1, ..., maxstep\}$ **do**
    $Logit, Value = Forward(Obs, Action)$ `//` forward propagation of neural networks, Obs is the observation (adjacency and feature matrix) of the current $TSSDN$
    $MaskedLogit = Apply\_Mask(Logit, Mask)$ `//` logit is $-\infty$ when masked by zero
    $a = Sampling(Action, MaskedLogit)$ `//` get the next action by sampling
    $TSSDN, Reward = Apply\_Action(TSSDN, a)$
    $G^f, ER = Failure\_Analysis(TSSDN, B, FS, R, \Phi)$ `//` by Algorithm 3 in Section V
    if $ER = \emptyset$ **then** `//` reliability requirement is met
      Record the best solution; reset $TSSDN$
    end
    $Action, Mask = SOAG(G^c, TSSDN, G^f, ER, K)$ `//` by Algorithm 1 in Section IV-B
    if $\forall m \in Mask, m = 0$ **then**
      Reset $TSSDN$; $Reward = Reward - 1$ `//` penalty for invalid solutions
    end
    Store $Obs, a, Value, Reward, Logit$ in RL Buffer
  end
  `// train neural networks with buffered data`
  Compute advantage estimation based on the value obtained
  Gradient ascent on GCN+actor MLP to maximize the PPO objective function
  Gradient descent on GCN+critic MLP to minimize the mean-squared error of the value function
end
return the best solution found

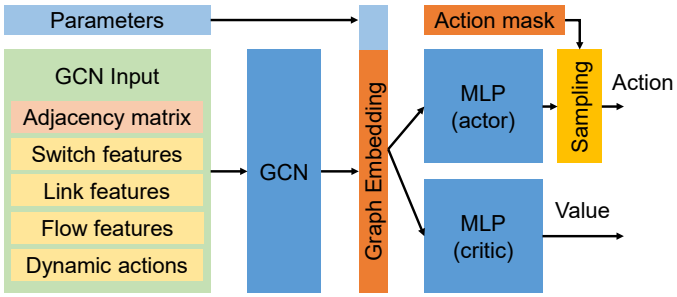


Fig. 3: Neural Network Architecture in the RL agent

the degree matrix of the self-connected adjacency matrix, i.e., $D_{i,i} = \sum_j (A_{i,j} + I_{i,j})$. W^l is the learnable weight at layer l.

$$H^{(l+1)} = \sigma\left(D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}H^l W^l\right) \quad (4)$$

TSSDN is naturally represented by graphs. Hence, NPTSN uses GCN to extract information from the graph representations of TSSDN. The neural networks in NPTSN are illustrated in Fig. 3. For TSSDN, the observation of the current state, which is the input of the GCN, consists of an adjacency matrix (A) and a list of node features (H). The node features are encoded into a $|V^c| \times x$ matrix in which x is the number of node features (x can be an arbitrary positive integer). The output of the GCN is a low-dimension graph embedding vector. To obtain the policy, the actor network takes the graph embedding as input and determines the logit probability from which the next action will be sampled. Parameters, such as flow periods and frame sizes, are not features of the graph. They can be represented by a vector for each flow. These vectors together with the vector of the network parameters (e.g., base period) are concatenated with the graph embedding vector as the input of the actor network. The critic network takes the same input as the actor network and produces a real number to estimate the value function of the selected actions. NPTSN uses Multi-Layer Perceptrons (MLP) for both the actor and critic networks

A possible alternative for GCN is the Graph Attention Networks (GAT). GAT introduces masked self-attentional layers to reduce the cost of assigning different weights to different neighboring nodes [22]. We do not select GAT because it has been outperformed by GCN in similar problems [16]. Besides, GAT is less scalable due to its massive memory consumption.

Encoding Method: to train the neural networks stably on a dynamic action space, both the network status and the dynamic actions should be encoded into the features of GCN, i.e., the states contain information regarding available actions. Thus, the GCN has four categories of input features as follows.

- **Switch features**: switch features are represented by a $|V^c| \times 1$ vector. Every element indicates the cost of a vertex. The switch cost is computed by $csw(deg(v), ASIL_v)$

while the end station cost is set to zero.

- **Link features**: link features are represented by a $|V^c| \times |V^c|$ matrix. Every element indicates the cost of the link (u, v) computed by $clk(ASIL_{u,v})$.
- **Flow features**: flow features are represented by a $|V^c| \times |V^c_{es}|$ matrix. Every element indicates the number of flow paths required by FS between $u \in V^c$ and $v \in V^c_{es}$. The element is set to zero if u is a switch.
  Note that counting only the number of paths loses information when flows have diverse parameters (e.g., period, frame sizes). A possible alternative is to represent the flow features with a $|V^c| \times |FS|$ matrix, where the source and destinations for each flow are represented by each column (e.g., source = 1, destination = 2, other vertices = zero). However, since there can be significantly more flows in a network than end stations, such encoding is less scalable.
- **Dynamic actions**: the actions are represented by a $|V^c| \times |K|$ matrix. An element is set to one if the corresponding vertex is traversed by the path, otherwise zero.

The overall feature matrix is the concatenation of the four feature matrices whose size is $|V^c| \times (1 + |V^c| + |V^c_{es}| + |K|)$.

The actor-critic algorithm: Algorithm 2 demonstrates the actor-critic algorithm [19] for training. In every epoch, the algorithm explores the current policy to gather a pre-defined number of steps, then updates the neural networks to improve the policy. In every step, the actor network determines the probability to select each action. The probability is filtered by the mask (line 6) using the technique in [16] to prevent invalid actions from being sampled. But the buffer must store the original policy without applying the mask (line 17) to ensure the correctness of the gradient computation. With the degree constraint enforced by the action mask, feasible solutions can be identified by checking the reliability goal only (line 9). If a solution is found, the TSSDN is reset for the next exploration as attempting more actions will further increase the network cost. If SOAG determines that no valid actions could be taken, the TSSDN is also reset to empty and the penalty is subtracted from the reward (line 15).

Since the dynamic actions are encoded into the feature matrix of Obs, training can be conducted with normal policy gradient methods for a static action space. The actor network is trained using the Proximal Policy Optimization (PPO), which is a state-of-the-art policy gradient method emulating monotonic gradient improvement [23], i.e., it attempts to update the gradient with the largest possible step while not causing performance collapse. The alternative to PPO is the Trust Region Policy Optimization (TRPO) [24], which guarantees the monotonic gradient improvement with a second-order method. Compared with TRPO, PPO reaches better sample complexity with first-order methods that is much simpler to implement [23]. The PPO objective is shown in Eq. 5, in which $r_t(\theta)$ is the probability ratio of the policy update. It is clipped into $[1-\epsilon, 1+\epsilon]$ to form a lower bound of the conservative policy iteration objective [25]. $\hat{A}_t$ is the advantage function. Our implementation uses the well-known Generalized Advantage Estimation (GAE)-Lambda advantage function [26].

Algorithm 3: Failure injection algorithm

input : topology G^t; specifications B, FS, R and Φ;
output: a non-recoverable failure and its error message
// compute max failure order
Sort the switches in V^t_{sw} by the reducing order of failure probability, i.e., $V^t_{sw} = \{v_1, v_2, ...\}$, $maxord$ is the maximum k so that $\prod_{i=1,...,k} cfp(ASIL_{v_k}) \geq R$
// check reliability requirement
$checked = \emptyset$
for $i \in \{maxord, ..., 1, 0\}$ **do**
  // check all subsets with i elements
  for $V^f \in combinations(V^t_{sw}, i)$ **do**
    if $\prod_{v \in V^f} cfp(ASIL_v) \geq R$ ***and*** $\forall\, V \in checked, V^f \not\subseteq V$ **then**
      // check recoverability
      $G^f = (V^f, \emptyset)$ // $E^f = \emptyset$
      $FI', ER = \Phi(G^t, G^f, B, FS)$
      if $ER \neq \emptyset$ **then**
        return G^f, ER // G^t invalid
      end
      $checked = checked \cup \{V^f\}$
    end
  end
end
// reliability requirement is met
return empty graph, $\emptyset$

$$L^{clip}(\theta) = \mathbb{E}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \quad (5)$$

The gradient descent on the critic network minimizes the mean-squared error of the value estimation, which is a common practice in actor-critic algorithms [19]. In total, the weights of the GCN are updated twice to improve the performance for both the actor and the critic networks. In our implementation, all gradient updates are performed with the well-known Adam gradient optimization algorithm [27].

A challenge to implementing Algorithm 2 is that the failure analysis is very expensive in terms of execution time. In this case, popular GPU acceleration techniques that mainly accelerate the neural networks bring limited benefit. Therefore, we parallelize Algorithm 2 using CPU which speeds up both the neural networks and the failure analysis. The idea is that the policy can be explored independently by each processor with the results stored in a local buffer (line 3-18). Given m processors, every processor only needs to explore $maxstep/m$ steps so the overall execution time can be reduced. To train the neural networks (line 19-21), the processors perform a distributed gradient computation to obtain a global gradient estimation. This involves calculating the average value of the gradient estimator over different processors [28]. With the same global gradient distributed to each processor, the processors perform a synchronized gradient update for their neural networks. It ensures that the policy is coherent for the next epoch.

V. Failure Analyzer

The failure analyzer injects failures into the TSSDN to evaluate its reliability. If the reliability guarantee is established, the solution can be recorded. Otherwise, the failure analyzer identifies a non-recoverable failure scenario for the SOAG to generate dynamic actions of the next iteration.

The failure analyzer supports arbitrary stateless NBF as its input. It is also based on a characteristic of the action space, i.e., the ASIL of every link (u, v) equals the lowest ASIL of the adjacent vertices u and v. Based on these conditions, the failure analyzer only checks the non-safe faults consisting of switch failures to ensure the reliability guarantee. In other words, although failures could occur with arbitrary links and switches, an exhaustive check of all possible faults is not necessary. The proof is as follows. Given an arbitrary failure G^f which is a non-safe fault containing link failures ($E^f \neq \emptyset$), it can be mapped to the failure scenario V'^f in Eq. 6 that only consists of the switch failures. Note that $low(u, w)$ in Eq. 6 returns the node with the lowest ASIL and, since end station failures are safe faults, G^f does not contain end stations.

$$V'^f = \{v | \exists (u, w) \in E^f, v = low(u, w) \vee v \in V^f\} \quad (6)$$

The ASIL of each link equals the lowest ASIL of the adjacent vertices. Thus, V'^f has a higher probability than G^f. Hence, V'^f is also a non-safe fault which the network must survive. Moreover, since a failed switch will disable all links attached to it, the residual network of failure V'^f is a subgraph of that of G^f. Thus, if the network can survive V'^f, the same flow state can also survive the failure of G^f (we assume the recovery mechanism is capable of finding it at run-time). Therefore, checking G^f is not necessary.

The failure analyzer executes the failure injection algorithm presented in Algorithm 3. It checks possible switch failures starting from the highest order (maximum number of components that fails, denoted by $maxord$). Since the flow state that survives failure V can also survive failure V^f if $V^f \subseteq V$, the failures checked are recorded to avoid checking their subsets again (line 11). The algorithm runs the specified NBF for failures with a probability larger than R and whose superset has not been checked. The algorithm involves running the NBF $O(|V^t_{sw}|^{maxord})$ times. Because $maxord$ depends on the failure rate of the switches, Algorithm 3 has exponential complexity (usually with a small exponent) if a polynomial time NBF, such as the recovery algorithm in [9], is given.

NPTSN can consider flow-level redundancy with a minor modification of the failure analysis (no need to change the intelligent network generator). In such cases, the NBF reports error messages when all redundant flow instances fail. The failure analysis then should check the possible failures for all network nodes to determine possible violations of the reliability goal (replacing all V^t_{sw} in algorithm 3 into V^t). Thus, the complexity of algorithm 3 becomes $O(|V^t|^{maxord})$.

VI. Evaluation

We implement NPTSN on a server equipped with an Intel core i9-9900K processor. The RL agent is implemented using PyTorch [29]. The training algorithm is based on the SpinningUp RL library [28]. It is paralleled over 8 cores which communicate with MPI. The SOAG and the failure analyzer are implemented with Python for simple integration.

TABLE II: NPTSN default RL parameters

Parameter	Value	Parameter	Value
Number of GCN layers	2	K	16
MLP hidden layers	256x256	maxepoch	256
Graph embedding features	$2 \times \lvert V^c \rvert$	maxstep	2048
Reward scaling factor	10^3	Clip ratio ϵ	0.2
Learning rate (actor)	3×10^{-4}	GAE Lambda	0.97
Learning rate (critic)	10^{-3}	Discount factor	0.99

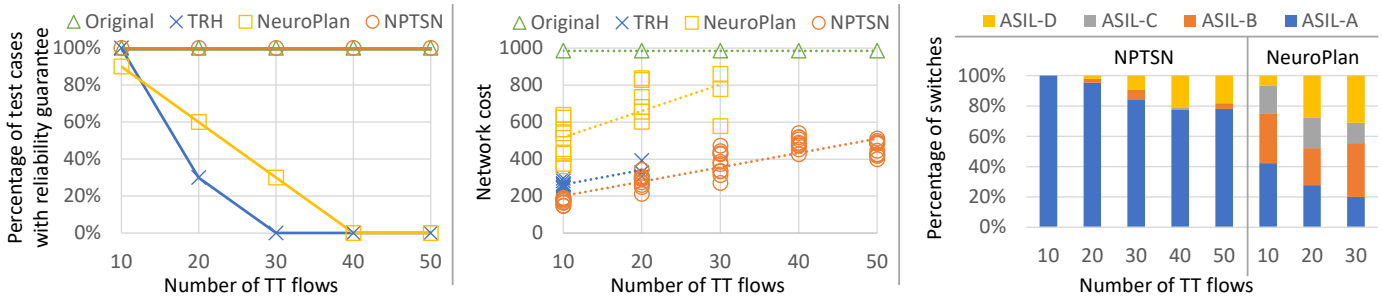
We evaluate NPTSN with two design scenarios: ORION [30] and ADS [31]. ORION is the design problem abstracted from the network architecture in the ORION crew exploration vehicle [30]. It is an aerospace network with a size larger than typical automotive TSSDN. Hence, we use it to demonstrate the solution quality and scalability of NPTSN. ADS is a design problem abstracted from an autonomous driving system [31]. It is realistic for IVN design. Thus, we use it for a sensitivity test which reveals the impact of different parameter settings. Note that our experiments for different design scenarios show similar trends. So, we only present the most suitable design scenarios for each evaluation for conciseness.

In all of the design scenarios, the NBF is selected to be the heuristic recovery algorithm in [9]. The maximum end station degree is set to 2, which is the minimum number to establish redundancy. Because the wiring distance in commercial vehicles is not available, we set the lengths of all optional links to be 1 unit length for simplicity. R is set to 10^{-6}, which is the minimum value that allows an ASIL-D device to function without a backup (so we can use the original ORION topology for baseline). The RL parameters are based on the default settings in the SpinningUp library, listed in TABLE II.

A. Performance Evaluation

The ORION design scenario requires planning a network with 31 end stations and 15 (optional) switches. We assume that a feasible link exists for any network node pairs within 3 hops in the original topology, resulting in 189 optional links inside E^c. The base period is set to 500 µs, which is uniformly divided into 20 time slots. Since the original ORION network hosts no more than 5 TT flows, we generate randomized flow specifications to test NPTSN with up to 50 flows. All flows are periodic unicast flows with period and deadline both being 500 µs. The sources and destinations are randomly selected from the end stations with uniform distribution. We generate test cases with 10, 20, 30, 40, and 50 flows, ten test cases for each number of flows, resulting in 5×10=50 test cases in total. Two metrics are evaluated: *the percentage of the test cases satisfying the reliability requirement* and *the cost of the best solution*. To the best of our knowledge, NPTSN is the first work that addresses the network planning problem for TSSDN protected by run-time recovery. Baselines for the exact same problem cannot be found in the literature. Thus, we compare NPTSN with three state-of-the-art solutions targeting similar problems.

- **The original network**: the manually designed topology for ORION can be found in [30]. We assign ASIL to its



(a). Reliability guarantee of different approaches (b). Solution quality of different approaches (c). ASIL allocation of different approaches

Fig. 4: Performance evaluation results

components for comparison with NPTSN solutions. In the original topology, since every end station is connected to one of the switches via a single link, single-point failures can completely isolate the end stations from the rest of the network. Hence, all network components require ASIL-D to ensure the reliability guarantee.

- **The TRH algorithm**: TRH is a heuristic algorithm proposed in [4] to synthesize topology for TSN with FRER protection. FRER provides reliability guarantees with static redundancy. It involves creating multiple disjoint FRER paths per flow and scheduling the flows simultaneously on their FRER paths. TRH does not consider ASIL but instead requires specifying the number of disjoint paths per flow. It adds the paths to the networks based on a breadth-first search algorithm. For comparison, we assign ASIL-B to all network components and use TRH to create two FRER paths per flow. Thus, the reliability guarantee is provided according to the ASIL decomposition rules [2]. Note that TRH does not consider schedulability. Instead, it is checked afterward to report invalid solutions.
- **NeuroPlan**: NeuroPlan is a state-of-the-art network planning tool for optical networks [16]. It first obtains initial solutions with RL and then optimizes the final results using Integer Linear Programming (ILP). Because it is impossible to model run-time recovery with ILP, we only adopt the RL stage as our baseline. The NeuroPlan agent chooses actions from a static action space to add capacity to different links without ASIL in its concern. For comparison, we modify the action into adding links and assigning switch ASIL. The link ASIL is determined by the ASIL of the adjacent switches same as in NPTSN. The NeuroPlan environment deals with the constraints of the optical networks. We modify it to reward the agent based on the network cost and the reliability same as NPTSN.

All solutions use the component library in TABLE I. Note that our component library is modified from the library in [4]. The original library does not consider ASIL and only contains switches with 3, 4, and 5 external ports. However, the ORION topology [30] requires switches with up to 8 ports. To make it suitable for all baselines, we assume that the ASIL-A switches have the same cost but 4, 6, and 8 ports. According to [32], increasing the ASIL by one level will increase the development cost by 1.25x-2x. Thus, we increase the switch (link) cost by 1.5x (2x) per ASIL. The failure probability is set according to the failure rate specified in ISO26262 [2]. We assume failures following the exponential distribution over 1000 working hours, e.g., the failure rate for ASIL-D is 10^{-9}, then the failure probability is $1 - e^{-10^{-9} * 1000} \approx 10^{-6}$.

In ORION test cases, the training of NPTSN takes approximately 39s per epoch, i.e., a test case takes around 2.7h without an early stop. The percentage of test cases with reliability guarantee for different numbers of TT flows is illustrated in Fig. 4(a). NPTSN guarantees reliability in all test cases. Given ASIL-D links and switches, the original ORION topology is also a valid solution for all test cases. However, for TRH and NeuroPlan, their capability to provide reliability guarantees significantly reduces when the network load increases. TRH cannot guarantee reliability for test cases with more than 20 flows. The main reason is that it does not consider schedulability, i.e., links can be shared by too many flows, which makes scheduling impossible. FRER redundancy doubles the network load, which aggravates the problem. NeuroPlan is not feasible for test cases with more than 30 flows mainly because its action design is not suitable for the sparse solution space. Although any topology can be constructed by adding individual links, it requires a long decision trajectory prone to mistakes. During the stochastic exploration, bad decisions could saturate the ports (degrees) of the switches preventing essential links from being added. As a result, the exploration frequently ends before solutions are found because switch degrees have been fully occupied. NPTSN does not suffer from these problems. Adding paths instead of links shortens the decision trajectory and the SOAG prunes away bad decisions. Thus, valid solutions are more likely to be found during stochastic exploration.

The network cost acquired by different approaches is illustrated in Fig. 4(b). The original topology always leads to the highest cost (i.e., 986) since it only uses ASIL-D components. In contrast, NPTSN achieves the lowest cost in all test cases. For instance, the minimum cost obtained by NPTSN is 146 for test cases with 10 flows, i.e., the cost achieved with NPTSN is up to 6.8x lower than the original network. TRH requires higher network costs to provide the same reliability as NPTSN. For instance, in the same test case where the cost by NPTSN is 146, the cost by TRH is 272 (1.8x). It is essentially because TRH

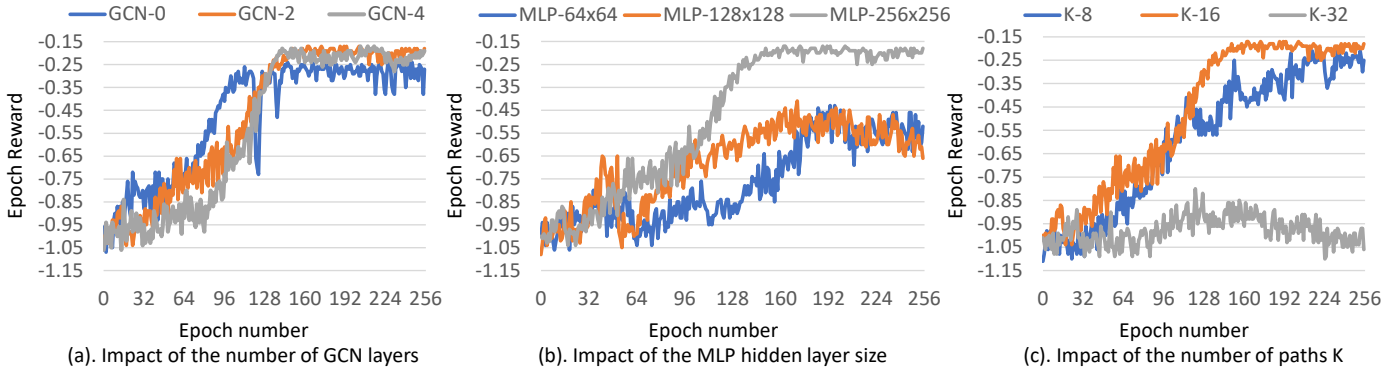


Fig. 5: Sensitivity test results

improves network reliability by replicating paths for individual flows. It does not consider how the added redundant links can be reused in different recovery scenarios (e.g., failures and flows). NPTSN instead performs more efficient optimization as it considers the reliability and cost impact of every path added through reward. NeuroPlan leads to higher costs even if it is modified to consider run-time recovery. This is again due to its long decision trajectory, since its RL agent makes stochastic decisions to add individual links. This causes a higher probability to introduce futile links. In NPTSN instead, paths are added targeting specific failure scenarios, which reduces the probability to introduce unnecessary connectivity.

Fig. 4(c) shows the distribution of the switches by their ASIL. Note that it only includes NPTSN and NeuroPlan because TRH and the original network allocate static ASIL to all switches (ASIL-B and ASIL-D). NPTSN incrementally upgrades switches to approach the solution from lower ASIL. In contrast, NeuroPlan explores the allocation of switch ASIL without search space pruning. It tends to use high-ASIL switches more frequently. For example, 31.1% of the switches are ASIL-D when there are 30 flows. This is an important reason for its high network cost. It implies that, at least with our setup, redundant networks with low ASIL components can be more cost-efficient than dense networks that require high ASIL components.

B. Sensitivity Test

The ADS design scenario involves planning the network for an autonomous driving system [31] using the component library in TABLE I, where NPTSN connects 12 end stations using a maximum of 4 switches. For ADS, G^c contains the complete set of connections, i.e., there are 54 optional links in E^c. Since the original flows are not available, we generate 12 flows using the same specification in Section VI-A based on the 7 safety-related applications onboard [31]. More precisely, there are two flows per application except for the vehicle state estimation which relies on other sensing applications to provide data. The optional links allow arbitrary connection except for the direct connection between end stations. we vary the three customized parameters (number of GCN layers, MLP hidden layer sizes, and K), one at a time, to demonstrate their impact on the training process. For each parameter, we plot the epoch reward versus the number of epochs. Note that here we only show results from one test case for simplicity. A similar trend can be observed in similar test cases according to our experiments. For ADS test cases, NPTSN takes approximately 10s per epoch. Thus, each test case requires around 40min.

NPTSN is tested with the number of GCN layers set to 0, 2, and 4. With the default learning rates, we observe that the neural networks without GCN (GCN-0) are unstable, i.e., the reward suddenly drops after a gradient update and never converges. Hence, for GCN-0, we adjust the actor learning rate to 1×10^{-4}. The reward versus the epoch number is shown in Fig. 5(a). The reward of GCN-0 varies between -0.25 and -0.4 after training. Instead, for GCN-2 and GCN-4, the reward converges around -0.2, because GCN can efficiently extract information from graph representations. According to our experiments, adding GCN layers causes a minor impact on performance and converging speed. In this case, both GCN-2 and GCN-4 achieve the same maximum reward of -0.17. And GCN-4 converges after 138 epochs, which is slightly faster than GCN-2 (epoch 149).

Fig. 5(b) shows the training process with MLP hidden size set to 64x64, 128x128, and 256x256. Generally, larger MLP leads to better training results because it can model more complex features. In this case, the reward of the 256x265 MLP converges around -0.2 while, for other setups, their rewards float around -0.55 with large variances. It can be observed that the size of MLP has a more significant impact on performance than the number of GCN layers.

Fig. 5(c) shows the experiment with various parameter K settings. Remember that K is the number of path addition actions generated during SOAG. Intuitively, more action to be selected indicates a larger coverage of the solution space. Thus, K-16 leads to faster convergence and a smoother reward curve than K-8. However, an overlarge K compromises SOAG in its capability to prune the search space. Thus, it can be difficult to discover good policies during exploration. Moreover, adding long paths to the networks is usually harmful because they involve adding more links and occupying more switch ports. As a result, even finding feasible solutions can be difficult. Therefore, as shown in the figure, the neural networks do not converge when K is set to 32.

VII. Related Work

This section introduces related works focusing on three aspects: run-time recovery of TSSDN, network planning with reliability concerns, and machine learning tools for TSN.

Run-time recovery: recovery mechanisms are studied in recent research to consider different network specifications and address different concerns. The switch-driven recovery in [14] restores flows in the real-time Ethernet with a bounded path restoration delay. But this approach cannot be directly applied to TSSDN where TT flows must be scheduled for TAS. [9] presents a mechanism to recover TT flows within 100ms against permanent failures considering the TAS scheduling problem. The mechanism in [8] instead deals with transient failures in TSN. [7] later proposes a recovery mechanism that offers protection for both transient and permanent failures. It restores replicated flow instances for FRER to maintain seamless redundancy. [6] studies the TSN recovery problem from a different perspective, i.e., it proposes a reconfiguration protocol so the recovery mechanisms above can safely deploy new configurations. All of the studies above are based on a common assumption. They expect the network topology to provide redundant paths so that a feasible configuration can be found to survive failures. Our work addresses this issue. NPTSN takes the recovery mechanism as input. It plans the TSSDN (topology and ASIL) to offer reliability guarantees.

Network planning: network planning is a well-known NP-hard problem to which TSSDN brings new challenges. Approaches for general network planning can be categorized into approximated algorithms and the exact algorithm [11]. Approximated algorithms, such as CEM [12], genetic algorithm [33], and dynamic programming [34], use well-developed optimization techniques to acquire reasonably good solutions. Instead, the exact algorithm always finds the optimal solution and outperforms the approximated algorithms [11]. However, general network planning approaches may compromise the reliability of TSSDN. TSSDN requires assigning ASIL, which is not considered for the general networks. Moreover, it requires the flows to be scheduled for TAS during the recovery process. Conventional networks instead deal with more relaxed latency requirements and usually do not involve such scheduling. They can be recovered if the topology after the failure remains connected. Thus, a reliable topology for conventional networks is not necessarily suitable for TSSDN.

Topology planning approaches [4], [35] have been studied for TSN with static FRER protection. However, they are hardcoded with techniques for static redundancy and cannot efficiently explore the recovery capability of TSSDN to perform cost reduction. These approaches essentially establish a pre-defined number of redundant paths and schedule flows statically on these paths. With FRER protection, the redundant paths can be designed according to the ASIL decomposition rules [2]. However, for TSSDN whose flow paths dynamically changed against failures, it leads to over-redundant networks with unnecessary costs. Moreover, some algorithms in [4] are based on ILP. But the dynamic recovery behaviors of TSSDN can not be modeled using linear constraints. Our work instead deals with the TSSDN planning problem leveraging RL tools. It allows directly representing the complex recovery behavior and the domain-specific requirements via the environment dynamics.

RL has recently been applied in planning optical networks [16]. They focus on different domain-specific constraints such as the spectrum consumption constraint. Both NPTSN and [16] utilize GCN and actor-critic algorithms, which are well-developed RL frameworks for solving graph-based problems. However, NPTSN combines the NBF-knowledge-based generation with neural network-based methods while the algorithm in [16] only uses neural network-based generation with static link-based actions. Thus, NPTSN performs active pruning and is more suitable for a sparse solution space. The dynamic actions also require novel methods to encode the actions into the observation of the RL agent to perform stable training.

Using machine learning for TSN: machine learning has recently demonstrated remarkable results in many fields. Here we focus on the machine learning approaches targeting TSN. Supervised learning has been applied to verify the TSN schedulability [36]. But for the TSSDN planning problem, supervised learning is not ideal since it requires a large data set to train the neural networks. Obtaining such a data set for IVN is impossible because a limited amount of IVN topologies have been built for vehicles and they often belong to different vendors. Generative Adversarial Network (GAN) [37] is another popular generative approach. It consists of two neural networks: a generator to produce counterfeits, and a discriminator to distinguish the counterfeits from the real data. GAN also has the same limitation in solving the TSSDN planning problem due to the lack of training data. In contrast, RL can perform automatic design space exploration and does not require a large data set for training. RL has been widely explored in solving combinatorial optimization problems, for which a survey can be found in [17], and demonstrated marvelous potential. In the field of TSN, RL is a popular approach to automate the design of the flow configurations [38]–[40], which is proved to be an NP-hard combinatorial optimization problem. NPTSN makes a unique contribution as it applies RL to design the TSSDN topology which guarantees the reliability for run-time recovery.

VIII. Conclusion

In this paper, we propose NPTSN that solves the TSSDN network planning problem with RL. NPTSN aims at providing reliability guarantees for the run-time recovery behavior while minimizing the network cost. It expresses the domain-specific requirements of TSSDN with the RL environment and performs an efficient design space exploration utilizing GCN. To handle the sparse solution space and simplify the reliability verification, we design a dynamic action space that offers coarse-grained actions generated to survive non-recoverable failures. To avoid invalid attempts, NPTSN generates action masks to filter the actions. The GCN-based neural network is trained with an actor-critic algorithm that handles the action masks and performs gradient optimization according to the PPO Objective. Our evaluation shows that NPTSN can provide reliability guarantees for more test cases while significantly reducing the network cost. The reason is that it actively prunes

the design space and uses actions that shorten the decision trajectory. Therefore, the probability to obtain invalid solutions or introduce unnecessary connectivity can be reduced significantly. NPTSN also has the advantage of approaching reliability goals with low ASIL components, which potentially reduces the network cost. In the future, we are interested in addressing other functional safety aspects regarding TSN topology design, such as protection against critical scenarios.

REFERENCES

[1] "ISO/IEC/IEEE international standard - local and metropolitan area networks - part 1q amendment 3: Enhancements for scheduled traffic," *ISO/IEC/IEEE 8802-1Q:2016/Amd.3:2017(E)*, pp. 1–62, 2018.

[2] "ISO standard 26262-1:2018 road vehicles-functional safety," *International Organization for Standardization*, 2011.

[3] "IEEE/ISO/IEC international standard - local and metropolitan area networks - part 1cb: Frame replication and elimination for reliability," *ISO/IEC/IEEE 8802-1CB:2019(E)*, pp. 1–106, 2019.

[4] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking," in *ACM RTNS 2017*, 2017, p. 267–276.

[5] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-Sensitive Software-Defined Network (TSSDN) for real-time applications," in *ACM RTNS 2016*, 2016, p. 193–202.

[6] A. Kostrzewa and R. Ernst, "Achieving safety and performance with reconfiguration protocol for Ethernet TSN in automotive systems," *Journal of Systems Architecture*, vol. 118, p. 102208, 2021.

[7] Z. Feng, Z. Gu, H. Yu, Q. Deng, and L. Niu, "Online re-routing and re-scheduling of time-triggered flows for fault tolerance in time-sensitive networking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.

[8] "Efficient reservation-based fault-tolerant scheduling for IEEE 802.1Qbv time-sensitive networking," *Journal of Systems Architecture*, vol. 123, p. 102381, 2022.

[9] W. Kong, M. Nabi, and K. Goossens, "Run-time recovery and failure analysis of time-triggered traffic in time sensitive networks," *IEEE Access*, vol. 9, pp. 91 710–91 722, 2021.

[10] J. Rushby, "Partitioning for safety and security: Requirements, mechanisms, and assurance," NASA Langley Research Center, NASA Contractor Report CR-1999-209347, Jun. 1999.

[11] M. Nishino, T. Inoue, N. Yasuda, S.-I. Minato, and M. Nagata, "Optimizing network reliability via best-first search over decision diagrams," in *IEEE INFOCOM 2018*, 2018, pp. 1817–1825.

[12] D. P. Kroese, K.-P. Hui, and S. Nariai, "Network reliability optimization via the cross-entropy method," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 275–287, 2007.

[13] "ISO/IEC/IEEE international standard - local and metropolitan area networks - part 1q amendment 31: Stream reservation protocol (SRP) enhancements and performance improvements," *IEEE Std 802.1Qcc-2018*, pp. 1–208, 2018.

[14] K. Lee, M. Kim, H. Kim, H. S. Chwa, J. Lee, and I. Shin, "Fault-resilient real-time communication using software-defined networking," in *IEEE RTAS 2019*, 2019, pp. 204–215.

[15] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NIPS 2018*, 2018, p. 6412–6422.

[16] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *ACM SIGCOMM 2021*, 2021, p. 258–271.

[17] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[18] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[19] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, 1999.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI 2018*, 2018.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML 2015*, 2015, pp. 1889–1897.

[25] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML 2002*, 2002, p. 267–274.

[26] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[30] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Syst.*, vol. 51, no. 1, p. 1–35, Jan. 2015.

[31] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of autonomous car-part II: A case study on the implementation of an autonomous driving system based on distributed architecture," *IEEE Trans. Ind. Electron.*, vol. 62, no. 8, pp. 5119–5132, 2015.

[32] G. Xie, Y. Chen, Y. Liu, R. Li, and K. Li, "Minimizing development cost with reliability goal for automotive functional safety during design phase," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 196–211, 2018.

[33] A. Kumar, R. Pathak, and Y. Gupta, "Genetic-algorithm-based reliability optimization for computer network expansion," *IEEE Transactions on Reliability*, vol. 44, no. 1, pp. 63–72, 1995.

[34] B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "Topology design with minimal cost subject to network reliability constraint," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 118–131, 2015.

[35] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for IEEE 802.1Qbv TSN networks," in *IEEE IOLTS 2018*, 2018, pp. 151–156.

[36] T. L. Mai, N. Navet, and J. Migge, "On the use of supervised machine learning for assessing schedulability: Application to Ethernet TSN," in *ACM RTNS 2019*, 2019, p. 143–153.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[38] L. Yang, Y. Wei, F. R. Yu, and Z. Han, "Joint routing and scheduling optimization in time-sensitive networks using graph convolutional network-based deep reinforcement learning," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[39] J. Prados-Garzon and T. Taleb, "Asynchronous time-sensitive networking for 5G backhauling," *IEEE Network*, vol. 35, no. 2, pp. 144–151, 2021.

[40] A. Grigorjew, M. Seufert, N. Wehner, J. Hofmann, and T. Hoßfeld, "ML-assisted latency assignments in time-sensitive networking," in *2021 IFIP/IEEE International Symposium on Integrated Network Management*, 2021, pp. 116–124.